Status Report: NAS Kernels in Fortran-8X and Optimized for the Connection Machine

Eric Barszcz*and Leigh Ann Tanner†

Report Number: RNR-90-018

National Aeronautics and
Space Administration

Ames Research Center
Moffett Field, California 94035

ARC 275 (Rev Feb 81)

Status Report: NAS Kernels in Fortran-8X and Optimized for the Connection Machine

Eric Barszcz*and Leigh Ann Tanner†

Report Number: RNR-90-018

January 30, 1990

Abstract

This is a report about the current status of the NAS Kernels running on the Cray Y-MP and Connection Machine CM-2 in Fortran-8X at NASA Ames Research Center. The processes of converting the kernels is presented and the problems encountered are described.

Keywords: Fortran-8X, CM-Fortran, Connection Machine, Benchmarking, NAS Kernels

*Applied Research Office, NAS Systems Division, NASA Ames Research Center, Moffett Field, CA. barszcz@orville.nas.nasa.gov

†An employee of Computer Sciences Corporation; this work was supported through NASA Contract No. NAS 2-12961, Systems Development Branch, NAS Systems Division, NASA Ames Research Center, Moffett Field, CA. tanner@sun224.nas.nasa.gov

1 Introduction

This is a report about the current status and results of running the NAS kernels in Fortran-8X[8] on a Cray Y-MP and a Connection Machine CM-2. The NAS kernel benchmark tests[1],[2] are a suite of seven computational kernels that are representative of numerically intensive codes used at NASA Ames Research Center. The original NAS kernels are written in Fortran77 (about 1000 lines). Both the Cray Y-MP and Connection Machine CM-2 used in the experiments are located at NASA Ames Research Center and will not be described in this report. For descriptions of the Cray Y-MP, see [7] and for the Connection Machine CM-2, see [5].

The version of Fortran that runs on the CM-2 is called CM-Fortran[3] and is a product of Thinking Machines Corporation. It provides the programmer with a subset of the proposed Fortran-8X language. It also includes much of Fortran77 with some extensions. If a Fortran program is run on the CM-2 without using any of the Fortran-8X construct provided in CM-Fortran, the program will be executed on the front-end machine without using the CM-2. The front-end machine at NASA Ames is a dual-processor VAX 6320, with 64 megabytes of main memory. Therefore, to run the NAS kernels on the CM-2 requires converting them to CM-Fortran 8X constructs.

When using the NAS kernels as benchmarks, not only is the hardware performance measured, but also the amount of tuning required to run them. Tuning ranges from level 0 ("dusty deck") to level 1000 ("customized code"). Since Fortran-8X array operations are required to execute code on the CM, it is considered customized code. Therefore, we are only interested in the performance that can be achieved while maintaining the functionality of the NAS kernels.

The cf77 compiler[6] on the Cray Y-MP also supports some of the Fortran-8X constructs, namely the array section constructs which are also a subset of CM-Fortran. To test the portability of code written in Fortran-8X, the kernels were converted to use array section constructs and run on the Y-MP and then ported to the CM-2. After initial porting to the CM-2, the algorithms are modified for efficiency.

In the next section, background on the NAS kernels is presented. Following that, the matrix multiply is used as an example of initial conversions from Fortran77 to Fortran-8X array sections. Then results from running on the Cray Y-MP are presented and discussed. Next, porting of Fortran codes to the Connection Machine CM-2 is discussed, followed by case by case experiences in converting and running each kernel. A discussion of results is given along with conclusions and suggestions for future work. An appendix is attached that contains the Fortran-8X version run on the Cray Y-MP. Code for the CM-2 is not included due to its rapidly changing state. Examples of it are throughout the text of this paper.

2 NAS Kernels

There are seven kernels in the NAS kernel benchmark suite. A brief description of each of the kernels follows:

- MXM: A four-way unrolled, outer product matrix multiply.
- CFFT2D: Complex radix 2 FFT on a two dimensional input array, returning results in place. FFT's for each dimension are handled by separate subroutines.
- CHOLSKY: Cholesky decomposition done in parallel on a set of input matrices that are passed in as one array.
- BTRIX: Block tridiagonal matrix solution along one dimension of a four dimensional array.
- GMTRY: Sets up arrays for a vortex code and performs Gaussian elimination on the resulting array.
- EMIT: Creates new vortices according to certain boundary conditions.
- VPENTA: Inverts three pentadiagonal matrices in a parallel fashion.

Tuning of the NAS kernels is broken down into four levels:

- Level 0, no changes are allowed for performance improvement, only for compatibility purposes (i.e. the timing function).
- Level 20, minor tuning to enhance performance is allowed (i.e. inserting compiler directives). No more than 20 lines may be changed.

- Level 50, major modifications to enhance performance is allowed (i.e. rewrite loops to avoid constructs that cause difficulties). No more than 50 lines may be inserted or changed.
- Level 1000, customized code to improve performance (i.e. rewriting entire subroutines). There is no limit on the number of lines inserted or modified.

The test suite has a main routine that calls each test individually. Each test generates its own input, makes a call to a system timer, performs the test for a specified number of iterations, makes a second call to the system timer, calculates an error value, the number of floating point operations performed and, how long the test took and, then reports the information back to the main routine. To generate input values, a random number generator is included in the suite that requires at least 47 bits of precision.

Errors are calculated based on having generated the correct random sequence and executed the test with sufficient precision. The number of floating point operations is based on the algorithm given in the source code and the number of iterations the test is performed. Timing is done by calling a system clock before and after the test and taking the difference. Performance is measured in MFLOPS.

3 Converting to 8X on Cray Y-MP

Converting the NAS kernels to use the 8X array section constructs was not difficult for most kernels. Conversion ranged from about 5 minutes for MXM (matrix multiply) to about 1.25 hours for EMIT (creation of vortices). The only 8X constructs supported by the cf77 compiler on the Y-MP are array assignment and array section constructs including vector valued subscripts.

An example of the use of the 8X array section construct is replacing

```
      DO   100 I = 1, 20
           C(I) = A(I) + B(I)
100   CONTINUE
```

by

```
C(1:20) = A(1:20) + B(1:20).
```

If the range is the whole dimension then an explicit range does not need to be specified, i.e. C(:) = A(:) + B(:). If whole arrays are being referenced then C = A + B is sufficient.

A vector valued subscript is the Fortran 8X notation for indirect array addressing. Say the elements of integer array IV are {1,3,5,2,4} and you want to access array A's elements in that order. The Fortran77 code would look like:

```
      DO   100 I = 1, 5
           C(I) = A(IV(I))
100   CONTINUE.
```

Using 8X constructs, the code becomes:

```
C(:) = A(IV(:)).
```

Often, the code is more compact using the 8X constructs. However, it can obscure what the code is doing if complex ranges are needed.

The general scheme used in converting from Fortran77 to Fortran-8X is to convert all in the vectorized innermost loops to array section constructs. The first and simplest kernel MXM (an outer product matrix multiply unrolled to a depth of four) is used as an example of this process. Fortran77 source code for MXM is given in figure 1. The converted code is shown in figure 2 and uses two of the 8X array constructs. Array sections are used to convert the innermost loop, reducing the levels of nesting, and array assignment is used to initial C to zero.

Results from running the unmodified Fortran77 version of the NAS kernels and from the Fortran-8X version are shown in table 1 and table 2 respectively. As can be seen from the tables, using the 8X constructs slow the Y-MP down by an average factor of 3.5 over the whole suite. However, some of the kernels run virtually unchanged (MXM and EMIT). Others such as CHOLSKY, BTRIX, and GMTRY have slowed down by a factor of 5 or more.

There is no apparent reason for the slow down. The only changes are the innermost vectorized loops. For example looking at CHOLSKY (Cholesky decomposition) shown in figure 3, one notes that all of the innermost loops are DO L = 0, NMAT where iterations of L are independent. Replacing these loops with array sections, (see figure 4), causes runtime to increase by a factor of 5. At this time, this phenomenon is

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      DIMENSION A(L,M), B(M,N), C(L,N)
C
C  4-WAY UNROLLED MATRIX MULTIPLY ROUTINE FOR VECTOR COMPUTERS.
C  M MUST BE A MULTIPLE OF 4.  CONTIGUOUS DATA ASSUMED.
C  D H BAILEY  11/15/84
C
      DO 100 K = 1, N
         DO 100 I = 1, L
            C(I,K) = 0.
100   CONTINUE
      DO 110 J = 1, M, 4
         DO 110 K = 1, N
            DO 110 I = 1, L
               C(I,K) = C(I,K) + A(I,J) * B(J,K)
     $                 + A(I,J+1) * B(J+1,K) + A(I,J+2) * B(J+2,K)
     $                 + A(I,J+3) * B(J+3,K)
110   CONTINUE
C
      RETURN
      END
```

Figure 1: Fortran77 version of MXM

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      DIMENSION A(L,M), B(M,N), C(L,N)
C
C  4-WAY UNROLLED MATRIX MULTIPLY ROUTINE FOR VECTOR COMPUTERS.
C  M MUST BE A MULTIPLE OF 4.  CONTIGUOUS DATA ASSUMED.
C  D H BAILEY  11/15/84
C  MODIFIED:
C    10/20/89  Eric Barszcz   Converted to Fortran-8X
C
      C = 0.0
      DO 110 J = 1, M, 4
         DO 110 K = 1, N
            C(:,K) = C(:,K) + A(:,J)   * B(J,K)
     $               + A(:,J+1) * B(J+1,K)
     $               + A(:,J+2) * B(J+2,K)
     $               + A(:,J+3) * B(J+3,K)
110   CONTINUE
C
      RETURN
      END
```

Figure 2: Fortran-8X version of MXM

THE NAS KERNEL BENCHMARK PROGRAM				
PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
MXM	1.8085E-13	4.1943E+08	1.5873	264.24
CFFT2D	3.1644E-12	4.9807E+08	7.4697	66.68
CHOLSKY	1.8256E-10	2.2103E+08	2.8060	78.77
BTRIX	6.0622E-12	3.2197E+08	2.4506	131.38
GMTRY	3.9145E-13	2.2650E+08	2.0594	109.98
EMIT	1.5609E-13	2.2604E+08	1.7639	128.15
VPENTA	2.3541E-13	2.5943E+08	5.3646	48.36
TOTAL	1.9275E-10	2.1725E+09	23.5015	92.44

Table 1: Fortran77 Version of NAS Kernels on Cray Y-MP

THE NAS KERNEL BENCHMARK PROGRAM				
PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
MXM	1.8085E-13	4.1943E+08	1.5785	265.72
CFFT2D	3.2001E-12	4.9807E+08	20.5480	24.24
CHOLSKY	1.8256E-10	2.2103E+08	14.5346	15.21
BTRIX	6.0622E-12	3.2197E+08	27.5411	11.69
GMTRY	6.5609E-13	2.2650E+08	10.6323	21.30
EMIT	1.5609E-13	2.2604E+08	1.7960	125.86
VPENTA	2.3541E-13	2.5943E+08	7.6205	34.04
TOTAL	1.9305E-10	2.1725E+09	84.2510	25.79

Table 2: Fortran-8X Version of NAS Kernels on Cray Y-MP

being turned over to Cray Research Incorporated. It is possible that the 8X constructs are inhibiting some optimization that is normally performed.

4 Converting to 8X on CM-2

Converting the 8X version of the NAS kernels developed on the Cray Y-MP to the Connection Machine CM-2 is not straight-forward. When the project started, the version of the CM-Fortran compiler available was Beta Version 0.5., which had a number of deficiencies. The major ones effecting the kernels are listed below. The original plan for the project was to test the kernels on the CM with the minimal amount of changes necessary to get them to run correctly. After that, make modifications to the algorithm for efficiency purposes. Unfortunately, even compiling them was a problem.

- Complex arithmetic (used in CFFT, GMTRY, and EMIT) is not supported.
- Declaration of arrays with negative indices (used in CHOLSKY) are not supported.

Other points to take into consideration while programming the CM are the following:

- Any COMMON containing a CM-resident array must be part of the MAIN routine.
- Routines are compiled completely independent of each other.
- The LAYOUT and ALIGN compiler directive are not fully functional.
- The compiler optimizes out multiple SUBROUTINE calls with empty parameter lists.

In order to have a calculation performed on the CM, the values must be contained in a CM resident array. There are three ways an array can become a CM resident array. First is to use the array in an 8X construct. This is easy and most robust. Second is to use the compiler directives LAYOUT or ALIGN. With LAYOUT, you specify whether each dimension is NEWS or SERIAL. NEWS implies it is spread along a NEWS axis (performed in parallel) in the machine. SERIAL implies the dimension is packed within a virtual processor (VP) set. Any array that has a dimension defined to be NEWS is a CM resident array. If all dimensions are declared to be SERIAL, then it is a front-end array. Version 0.5 of the compiler does not support mixed NEWs and SERIAL LAYOUT directives. The third way is by using COMMON. Any COMMON that contains a CM array must be included in the MAIN routine when it is being compiled. Space for the array is allocated on both the CM and the front-end. CM-Fortran does not currently support automatic array migration between the front-end and CM. There are compiler directives that specify that a named COMMON is a CM only COMMON to prevent allocation of space on the front-end.

How the location of the array effects the programmer is when passing arrays to subroutines. If the calling routine has the array defined as a CM array but the called routine never uses the array in an 8X construct then the compiler treats it as a front-end array with no compiler warnings. When the code is executed, it will abort. An example where this could happen is in figure 2. Assume arrays A, B and, C are all CM resident arrays in the calling routine. When MXM is called, array B will be considered a front-end array by the compiler because it does not satisfy any of the three conditions to be considered a CM resident array. Recall each routine is compiled independently.

Part way through the project, Beta Version 0.6 of the CM-Fortran compiler arrived. It is more robust than the Version 0.5 and supports complex arithmetic and mixed NEWS/SERIAL LAYOUT directives. Other problems exist that are related to the global optimizer:

- Multiple function calls with empty parameter lists are optimized out (erroneously reported as fixed in Release Notes Version 5.2-0.6[4].
- Complex arithmetic gives poor or incorrect results.
- Compiler switch '-double_precision' causes some fields to be overwritten.

Other bugs have been reported to Thinking Machines Corporation (TMC) and are still unresolved. Bugs pertaining to individual kernels will be discussed on a kernel by kernel basis. In the following sections, each kernel is discussed describing what was done, bugs encountered and known workarounds.

```
      SUBROUTINE CHOLSKY (IDA, NMAT, M, N, A, NRHS, IDB, B)
C
C  CHOLESKY DECOMPOSITION/SUBSTITUTION SUBROUTINE.
C
C  11/28/84  D H BAILEY  MODIFIED FOR NAS KERNEL TEST
C
      REAL A(0:IDA, -M:0, 0:N), B(0:NRHS, 0:IDB, 0:N), EPSS(0:256)
      DATA EPS/1E-13/
C
C CHOLESKY DECOMPOSITION
C
      DO 1 J = 0, N
         I0 = MAX ( -M, -J )
C
C OFF DIAGONAL ELEMENTS
C
         DO 2 I = I0, -1
            DO 3 JJ = I0 - I, -1
               DO 3 L = 0, NMAT
3                 A(L,I,J) = A(L,I,J) - A(L,JJ,I+J) * A(L,I+JJ,J)
            DO 2 L = 0, NMAT
2              A(L,I,J) = A(L,I,J) * A(L,0,I+J)
C
C STORE INVERSE OF DIAGONAL ELEMENTS
C
         DO 4 L = 0, NMAT
4           EPSS(L) = EPS * A(L,0,J)
         DO 5 JJ = I0, -1
            DO 5 L = 0, NMAT
5              A(L,0,J) = A(L,0,J) - A(L,JJ,J) ** 2
         DO 1 L = 0, NMAT
1           A(L,0,J) = 1. / SQRT ( ABS (EPSS(L) + A(L,0,J)) )
C
C SOLUTION
C
      DO 6 I = 0, NRHS
         DO 7 K = 0, N
            DO 8 L = 0, NMAT
8              B(I,L,K) = B(I,L,K) * A(L,0,K)
            DO 7 JJ = 1, MIN (M, N-K)
               DO 7 L = 0, NMAT
7                 B(I,L,K+JJ) = B(I,L,K+JJ) -  A(L,-JJ,K+JJ) * B(I,L,K)
C
         DO 6 K = N, 0, -1
            DO 9 L = 0, NMAT
9              B(I,L,K) = B(I,L,K) * A(L,0,K)
            DO 6 JJ = 1, MIN (M, K)
               DO 6 L = 0, NMAT
6                 B(I,L,K-JJ) = B(I,L,K-JJ) - A(L,-JJ,K) * B(I,L,K)
C
      RETURN
      END
```

Figure 3: Fortran77 version of CHOLSKY

```
      SUBROUTINE CHOLSKY (IDA, NMAT, M, N, A, NRHS, IDB, B)
C
C  CHOLESKY DECOMPOSITION/SUBSTITUTION SUBROUTINE.
C
C  11/28/84  D H BAILEY  MODIFIED FOR NAS KERNEL TEST
C  MODIFIED:
C    10/29/89  Eric Barszcz  Converted to Fortran-8X
C
      REAL A(0:IDA, -M:0, 0:N), B(0:NRHS, 0:IDB, 0:N), EPSS(0:256)
      DATA EPS/1E-13/
C
C  CHOLESKY DECOMPOSITION
C
      DO 1 J = 0, N
         I0 = MAX ( -M, -J )
C
C  OFF DIAGONAL ELEMENTS
C
         DO 2 I = I0, -1
            DO 3 JJ = I0 - I, -1
               A(:,I,J) = A(:,I,J) - A(:,JJ,I+J) * A(:,I+JJ,J)
3           CONTINUE
            A(:,I,J) = A(:,I,J) * A(:,0,I+J)
2        CONTINUE
C
C  STORE INVERSE OF DIAGONAL ELEMENTS
C
         EPSS = EPS * A(:,0,J)
         DO 5 JJ = I0, -1
            A(:,0,J) = A(:,0,J) - A(:,JJ,J) ** 2
5        CONTINUE
         A(:,0,J) = 1. / SQRT ( ABS (EPSS + A(:,0,J)) )
1     CONTINUE
C
C  SOLUTION
C
      DO 6 I = 0, NRHS
         DO 7 K = 0, N
            B(I,:,K) = B(I,:,K) * A(:,0,K)
            DO 7 JJ = 1, MIN (M, N-K)
               B(I,:,K+JJ) = B(I,:,K+JJ) -  A(:,-JJ,K+JJ) * B(I,:,K)
7        CONTINUE
C
         DO 6 K = N, 0, -1
            B(I,:,K) = B(I,:,K) * A(:,0,K)
            DO 6 JJ = 1, MIN (M, K)
               B(I,:,K-JJ) = B(I,:,K-JJ) - A(:,-JJ,K) * B(I,:,K)
6     CONTINUE
C
      RETURN
      END
```

Figure 4: Fortran-8X version of CHOLSKY

THE NAS KERNEL BENCHMARK PROGRAM				
PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
OUTER	1.1650E-06	4.1943E+06	31.5500	0.13
INNER	0.0000E+00	4.1943E+06	64.8200	0.06
SPREAD	0.0000E+00	4.1943E+06	2.4300	1.73
SQUARE	5.2117E-06	2.6844E+08	25.5200	10.52
NO OPT	4.4587E-06	2.6844E+08	8.3100	32.30
OPTIMIZE	5.2117E-06	2.6844E+08	2.4600	109.12
TMC's	4.4587E-06	2.6844E+08	1.3800	194.52

Table 3: Run times for MXM on 8K CM-2 in single precision.

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      DIMENSION A(L,M), B(M,N), C(L,N)
CMF$ LAYOUT B(:NEWS, :NEWS)
C
      C = 0.0
      DO 110 J = 1, M, 4
         DO 110 K = 1, N
            C(:,K) = C(:,K) + A(:,J)   * B(J,K)
     $                + A(:,J+1) * B(J+1,K)
     $                + A(:,J+2) * B(J+2,K)
     $                + A(:,J+3) * B(J+3,K)
110   CONTINUE
C
      RETURN
      END
```

Figure 5: Outer Product Fortran-8X version of MXM

4.1 MXM

The first NAS kernel studied on the CM was the matrix multiply. It is a simple problem with $O(n^3)$ parallelism if all multiply operations are done in parallel. Restricting ourselves to $O(n^2)$ memory implies there must be at least one level of iteration. If one starts by compiling the 8X version developed on the Y-MP, the problems with COMMON and the classification of array B immediately surface. Advancing past those problems, the code will compile and execute correctly.

Table 3 contains execution times for various algorithms to perform the matrix multiply. All timings were done using 8K processors of the CM-2 using single precision. The first three kernels perform one iteration of a 256x128 times 128x64 matrix multiply. The last three do one iteration of a 512x512 times 512x512 matrix multiply.

OUTER is the four way unrolled outer product matrix multiply taken from the Y-MP, figure 5. INNER is the standard inner product form performing one inner product at a time, figure 6. SPREAD performs all inner products for a row at a time, figure 7. SQUARE uses the systolic algorithm for matrix multiply based on the source provided with the release of the CM Fortran compiler, figure 8. NO OPT differs from SQUARE by using ALIGN directive to overlay arrays B and C on top of A, figure 9. It is compiled without the global optimization flag. OPTIMIZE is the same as NO OPT but compiled with the global optimization flag turned on, figure 9. The dramatic effect of the optimizer is seen. TMC's is the low level matrix multiply library routine provided by TMC that can be accessed from CM-Fortran.

The algorithm used in SQUARE, NO OPT, and OPTIMIZE only works on square arrays. It have been extended to work on rectangular arrays by the author and has the same performance on square arrays.

As can be seen from the results, the CM is extremely algorithm sensitive. There are three orders of magnitude between INNER and OPTIMIZE. It also implies that simply porting algorithms used for vector

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      REAL A(L,M), B(M,N), C(L,N)
C
      C = 0.0
      DO 110 I = 1, L
          DO 110 J = 1, N
              C(I,J) = SUM((A(I,:) * B(:,J)), 1)
110   CONTINUE
C
      RETURN
      END
```

Figure 6: Inner Product Fortran-8X version of MXM

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      REAL A(L,M), B(M,N), C(L,N)
      REAL TMP(M,N)
C
      DO 110 I = 1, L
          TMP = SPREAD(A(I,:), DIM=2, NCOPIES=N)
          C(I,:) = SUM((TMP * B), DIM=1)
110   CONTINUE
C
      RETURN
      END
```

Figure 7: Parallel Inner Products Fortran-8X version of MXM

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      REAL A(L,M), B(M,N), C(L,N)
      INTEGER Forward(L), Back(L)
C
      CALL CMF_COORDINATE(Forward,1)
      Forward = Forward - 1
      Back = -Forward
C
C Skew matrices so Diagonal(A) is in Column 1 and
C           Diagonal(B) is in Row 1
C
      A = CSHIFT(A, DIM=2, SHIFT=Forward)
      B = CSHIFT(B, DIM=1, SHIFT=Forward)
C
C Perform matrix multiplication.
C
      C = 0.0
      DO I=1,M
          C = C + A * B

          A = CSHIFT(A,DIM=2,SHIFT=1)
          B = CSHIFT(B,DIM=1,SHIFT=1)
      END DO
C
C Shift Matrices back to original positions
C
      A = CSHIFT(A, DIM=2, SHIFT=Back)
      B = CSHIFT(B, DIM=1, SHIFT=Back)
C
      RETURN
      END
```

Figure 8: Systolic Matrix Multiply Fortran-8X version of MXM

```
      SUBROUTINE MXM (A, B, C, L, M, N)
      REAL A(L,M), B(M,N), C(L,N)
      INTEGER Forward(L), Back(L)
CMF$ ALIGN B(I,J) WITH A(I,J)
CMF$ ALIGN C(I,J) WITH A(I,J)
C
      CALL CMF_COORDINATE(Forward,1)
      Forward = Forward - 1
      Back = -Forward
C
C Skew matrices so Diagonal(A) is in Column 1 and
C          Diagonal(B) is in Row 1
C
      A = CSHIFT(A, DIM=2, SHIFT=Forward)
      B = CSHIFT(B, DIM=1, SHIFT=Forward)
C
C Perform matrix multiplication.
C
      C = 0.0
      DO I=1,M
         C = C + A * B
C
         A = CSHIFT(A,DIM=2,SHIFT=1)
         B = CSHIFT(B,DIM=1,SHIFT=1)
      END DO
C
C Shift Matrices back to original positions
C
      A = CSHIFT(A, DIM=2, SHIFT=Back)
      B = CSHIFT(B, DIM=1, SHIFT=Back)
C
      RETURN
      END
```

Figure 9: Optimized Systolic Matrix Multiply Fortran-8X version of MXM

THE NAS KERNEL BENCHMARK PROGRAM				
PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
REAL	1.2566E-13	4.9807E+06	153.5800	0.03
COMPLEX	4.8710E-08	4.9807E+06	180.5400	0.03

Table 4: Run times for CFFT on 8K CM-2 in double precision.

THE NAS KERNEL BENCHMARK PROGRAM				
PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
GOOD CHO	4.2092E-12	1.1052E+06	905.5500	0.00
BAD CHO	1.8987E+02	1.1052E+06	783.6900	0.00

Table 5: Run times for CHOLSKY on 8K CM-2 in double precision.

machines probably will not do well on the CM. Vector machines can be thought of as having one dimensional parallelism. The CM requires two or three dimensional parallelism to run efficiently or one dimensional parallelism on the order of the number of processors.

One of the common tricks used in Fortran to generate longer vectors is to redeclare the shape of the array between subroutines. This is not possible on the CM. An array must have the same shape in all subroutines. However, the parallelism inherent in the algorithm can be expressed as two dimensional parallelism rather than one dimensional. There are functions that allow the shape of an array to be changed but they are not supported at this time.

4.2 CFFT

CFFT is a two dimensional FFT where each dimension is handled by a separate routine. Table 4 contains information from executing the FFT using real and complex arithmetic. Again, the inner loops are converted to array section constructs.

With the Version 0.5 of the CM-Fortran compiler, complex arithmetic is not supported. To run the CFFT test, all complex calculations are done out in real arithmetic. It has a poor run time but has a good error value. In fact, it is better that the Y-MP's. The poor run time is largely due to performing the FFT in double precision arithmetic. The CM located at Ames has only single precision hardware causing double precision arithmetic to be performed in software. Using Version 0.6 of the compiler which supports complex arithmetic, the error is five orders of magnitude worse than when performed using real arithmetic. The problem was reported to TMC. It appears the use of the global optimization flag causes a portion of the complex field to be overwritten. The workaround is to not use the global optimization flag. This can seriously effect the run time as demonstrated by the matrix multiply.

4.3 CHOLSKY

Using Version 0.5 of the compiler, CHOLSKY would not compile and run because arrays are declared with negative index ranges. This is still a problem with Version 0.6 of the compiler.

After converting to positive index ranges, the code compiles and executes but yields a bad error value. Table 5 shows the difference before and after the last conversion. Figure 10 contains the code for CHOLSKY that is causing the problem. It is currently being investigated at TMC.

4.4 BTRIX

BTRIX was compiled and run. It ran for 1.5 hours (performing one iteration) and then aborted with a software timeout. As of the writing of this report, the cause has not been investigated.

```
      SUBROUTINE CHOLSKY (IDA, NMAT, M, N, A, NRHS, IDB, B)
C
C  CHOLESKY DECOMPOSITION/SUBSTITUTION SUBROUTINE.
C
C  11/28/84  D H BAILEY  MODIFIED FOR NAS KERNEL TEST
C  MODIFIED:
C    11/20/89  Eric Barszcz  Converted to Frotran 8X
C
      DOUBLE PRECISION A(IDA, M, N), B(NRHS, IDB, N),
     $     EPSS(IDA), EPS
      DATA EPS/1D-13/
C
C CHOLESKY DECOMPOSITION
C
      DO 1 J = 1, N
         I0 = MIN ( M, J )
C
C OFF DIAGONAL ELEMENTS
C
         DO 2 I = I0, 2, -1
            I0I1 = I0 - I + 1
            DO 3 JJ = I0I1, 2, -1
CEB *****
CEB These lines generate Correct Results
CEB NOTE: NMAT iterations are Independent
                  DO 3 L = 1, NMAT
                     A(L,I,J) = A(L,I,J) - A(L,JJ,J-I+1) * A(L,JJ+I-1,J)
CEB *****
CEB This line generates INCORRECT Results
C                 A(:,I,J) = A(:,I,J) - A(:,JJ,J-I+1) * A(:,JJ+I-1,J)
CEB *****
3           CONTINUE
2        A(:,I,J) = A(:,I,J) * A(:,1,J-I+1)
C
C STORE INVERSE OF DIAGONAL ELEMENTS
C
         EPSS = EPS * A(:,1,J)
         DO 5 JJ = I0, 2, -1
5           A(:,1,J) = A(:,1,J) - A(:,JJ,J) ** 2
1     A(:,1,J) = 1.D0 / SQRT (ABS (EPSS + A(:,1,J)))
C
C SOLUTION
C
      DO 6 I = 1, NRHS
         DO 7 K = 1, N
            B(I,:,K) = B(I,:,K) * A(:,1,K)
            DO 7 JJ = 2, MIN (M, N-K+1)
               DO 7 L = 1, NMAT
7                 B(I,L,K+JJ-1) = B(I,L,K+JJ-1) - A(L,JJ,K+JJ-1) * B(I,L,K)
C
         DO 6 K = N, 1, -1
            B(I,:,K) = B(I,:,K) * A(:,1,K)
            DO 6 JJ = 2, MIN (M, K)
               DO 6 L = 1, NMAT
6                 B(I,L,K-JJ+1) = B(I,L,K-JJ+1) - A(L,JJ,K) * B(I,L,K)
C
      RETURN
      END
```

Figure 10: Problem with Fortran-8X version of CHOLSKY on CM.

THE NAS KERNEL BENCHMARK PROGRAM				
PROGRAM	ERROR	FP OPS	SECONDS	MFLOPS
VPENTA	2.5078E+02	6.4858E+05	10.4400	0.06

Table 6: Run times for VPENTA on 8K CM-2 in double precision.

4.5 GMTRY

GMTRY compiles and aborts during execution. As of the writing of this report, the cause has not been investigated.

4.6 EMIT

EMIT compiles and aborts during execution. As of the writing of this report, the cause has not been investigated.

4.7 VPENTA

VPENTA compiles and runs. Results are given in table 6. It completes but gives incorrect results. As of the writing of this report, the cause has not been investigated.

5 Conclusions

Results to date indicate the CM-Fortran compiler is not stable and that the Connection Machine is extremely algorithm sensitive. Most of the problems have been reported to TMC and they are actively seeking solutions. The sensitivity of the architecture to the algorithm and lack of common programming language across multiprocessors lead to the conclusion that NAS kernel style benchmarking is not possible for parallel computers. One possible approach to benchmarking parallel computers is to specify the desired functionality and remove any code modification restrictions or penalties. However, it is necessary to require that the code be written in a specified high level language and that the source be made available for verification and use. Along with the source, an explanation of algorithm choice should also be mandatory. This would provide insight into programming the machine as well as an efficient algorithm.

6 Acknowledgements

The authors would like to acknowledge the help of Kyra Lowther and Jacek Myczkowski of Thinking Machines Corporation for being responsive to problems and questions.

References

[1] Bailey, D.H., and Barton, J.T., "The NAS Kernel Benchmark Program", *NASA Technical Memorandum 86711*, August 1985.

[2] Bailey, D.H., "NAS Kernel Benchmark Results", *Frist International Conference on Supercomputing Systems*, IEEE Computer Society, 1985, 341-345.

[3] "CM Fortran Reference Manual",*Version 5.2-0.6*, Thinking Machines Corporation, Cambridge, Massachusetts, September 1989.

[4] "CM Fortran Release Notes",*Version 5.2-0.6*, Thinking Machines Corporation, Cambridge, Massachusetts, September 1989.

[5] Hillis, D.H., "The Connection Machine", *The MIT Press*, Cambridge, Massachusetts, 1985.

[6] "CFT77 Reference Manual",*SR-0018 C*, Cray Reasearch, Inc., Mendota Heights, MN, October 1988.

[7] "Training Workbook CFT77 on CRAY X-MP and CRAY Y-MP Computer Systems",*Revision B.02*, Cray Reasearch, Inc., Mendota Heights, MN, June 1988.

[8] Latest Fortran 8-X Draft Standard. CHECK ON THIS!!!!!!